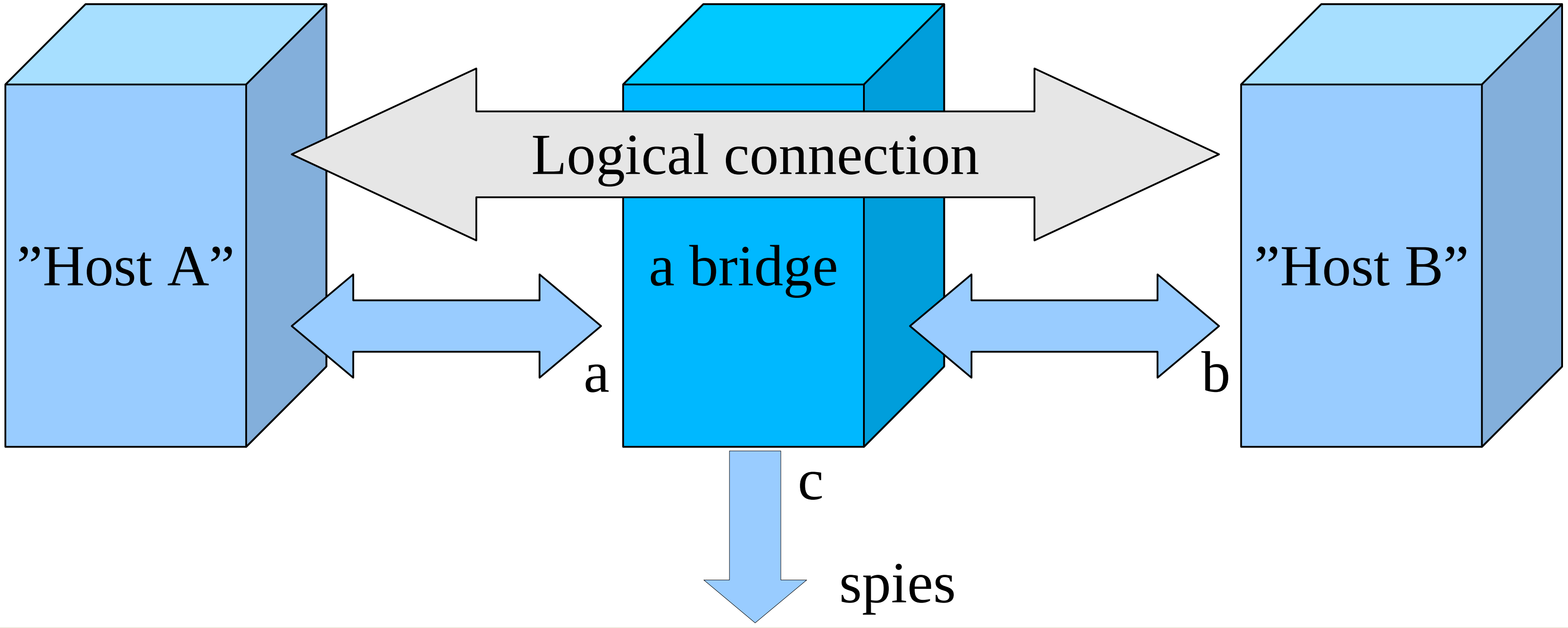
Assignment 1: tcpbridge

The assignment is to create a program that forwards the data from the bridge port (a) to other host's port (b). The target host's address details and port numbers to be used are specified as command line options. The bridge also listens for connections on port c and echoes all the traffic transmitted between port a and port b to clients connected on port c

There could be more than one client connected to port c.



tcpbridge

- **Listens for TCP connection request on a specified port (a)**
- **When the client connects, the program starts receiving data from the connected client and makes a connection to host B, with which the traffic would be exchanged. The bridge also has to start listening on another port(c) for incoming connection request.**
- **The data tranmitted between the client on port ‘a‘ and the host on port port ‘b ‘ is forwarded to all clients connected on the port c**
- **Short and hex form can be used for logging the data transmitted trought the bridge**
- **Terminating the program with Ctrl-C (SIGINT) will cause it to dump a summary on the number of bytes received so far.**
- **The program need to run for a specified time period**.

tcpbridge -i <port-no> -o <port-no> -s -l <dumplen> -d <dest> -t <duration>

-i:	**Open a listening socket for incoming TCP connection. The program needs to receive data from the client that connects to this port.**
-o:	**Open a listening socket for incoming TCP connection. Every client that connects to this TCP port number will receive a copy of the output, until it disconnects.**
	(Note: The format of the output need to be decided as,
	-s -> short form of output
	-l -> hex dump format
	If both -s and -l are not specified, then transfer the output, without any formatting)
-h:	**the destination hostname where the connection has to be “forwarded”**
-d:	**the destination IP address where the connection has to be “forwarded”**
-s:	**(Short form) Creates a single line of output for the data read**
	(<80 characters), containing a timestamp with microsecond resolution, source and destination IP address and port number and the number of bytes read or received.

tcpbridge contd…

-l: **Turns on the the hex dump and sets the hex output length to the specified number of bytes per send operation. Append the short form output before the hex dump. So that we could read the address and port information of the machines that exchange data.**

-t: **This option is used to specify a duration that the program shall run for. After this time is elapsed the program shall terminate automatically. The time interval shall be specified (as an integer) in seconds. A summary information on the number of bytes exchanged between either side, need to be printed to the stdout.**

Examples:

```
tcpbridge -i 5000 -o 5050 -h abcd.hut.fi:6000 -l 400 -t 200
tcpbridge -i 5000 -o 5050 -h abcd.hut.fi:6000 -s -t 200
tcpbridge -i 5000 -o 5050 -d 130.233.11.11:6000 -l 400 -t 200
tcpbridge -i 5000 -o 5050 -d 130.233.11.11:6000 -s -t 200
```

The assignment requirements expect support for IPv4 addressing.

Optional: Would be nice, if support can be provided for IPv6 too.

tester_utility

- A Utility would be provided to aid in the develpment and testing of the tcpbridge. A binary compiled under unix gcc would be provided.
- Note: The test program could be used make few test connections to specified addresses, but they are not used for testing the complete tcpbridge implementation.
- The utility operates in two modes

  send mode: It makes a TCP connection to a specified address and once a connection is established, it starts sending some data.

  receive mode: It makes a TCP connection to a specified address and once a connection is established, it starts receiving the data and dumps it to the stdout, if specifed (command line)

Test utility usage

- -s: To run the program in send mode
- -r: To run the program in receive mode
- -i: IP Address to which the connection need to be made
- -h: HOST Name to which the connection need to be made
- -p: Port Number to which the connection need to be made
- -n: Number of packets to be sent- one packet is sent per second
- (Used only by the sender mode)
- (Here – packet means some junk of data with no header)
- -d: Duration - The time for which the receiver has to receive data
- (Used only by the receiver mode)
- -l: Length - The length of the message to be printed
- (Used only by the receiver mode and for non binary data
- Note: Do not use '-l' if data that is to be received is binary)

Hints (1)

- Accepting a TCP connection
  - Use listening socket socket (SOCK_STREAM, AF_INET, …)
  - accept () incoming connections for spies
    - If there is more than one connected to spy the exchages, then, distribute the short form or hex dump form of output data to all of them
  - A select() READ event and a subsequent recv() result of “0” or “-1” indicate that the spy is gone.
    - Try this out, this varies between different operating systems
    - Close the socket locally
    - If none are left, revert to dumping to the screen
  - For this assignment, use all sockets in blocking mode (default setting)
    - Using non-blocking i/o will make things far too complicated and cause extra headache
    - But would, nevertheless, be the right thing to do in practice
  - Test with telnet(1)
    - telnet 127.0.0.1 50000   if your process listens on port #50000
    - you could also use the test utility for testing the listening ports

Hints (2)

- Time handling
  - gettimeofday yields detailed system clock reading as (sec, usec) pair
  - In the mainloop, determine the time to wait based upon the current time
    - This result is what you feed into poll() or select()
    - Note that both use completely different time formats
  - If select() returns 0, a timeout has occurred
- **Single line format, e.g.:**
- (when 's' switch is enabled use this format to represent the output)

```
14:09:00.123456 134.102.218.59:40000 -> 134.102.218.58:47000 79 Bytes
```

  - **Remember again network vs. host byte order**
- **Sample hex dump (when 'l' switch is used)**

```
Offset    Individual bytes in hexadecimal                   Characters (*)
--------- ------------------------------------------------  ------------------
000000: 00 01 02 00 41 42 43 09 00 64 00 00 00 00 30 39    ....ABC..d....09
000010: 00 00 0a 0d
```

Hints (3)

- **Do not use signals for timing**
  - Such as done by alarm(). This may just cause system call interruptions that you do not want or need
- Signals
  - Need to catch at least SIGINT: signal (SIGINT, signalhandler);
    - This may occur at any point in time, so you may want to postpone processing to the main loop (probably not needed in our simple example)
    - In this case, you would just set a global variable and return (terminate = 1;)
    - Need to check the variable regularly even if no data arrive
  - Will cause interrupted system calls (errno == EINTR)
    - Need to check for this also in your main loop and behave accordingly
- Short note on hexdumps()
  - printf (“%02x”, variable) prints the contents as 2 hex digits with leading zero
  - Exception: if the the highest bit is “1”, then leading “ffffff” may appear.
  - Solution: use (variable & 0x0ff) for printing (you care about 8 bits only)

Hints (4)

```
/* command line processing goes here */

if ((s = socket (AF_INET, SOCK_STREAM, 0)) == -1) {
   perror ("cannot create socket");
   exit (-1);
}
listen_addr.sin_family      = AF_INET;
listen_addr.sin_addr.s_addr = INADDR_ANY;
listen_addr.sin_port        = htons (listen_port);
if (bind (s, (struct sockaddr *) &listen_addr,
    sizeof (listen_addr)) == -1) {
    perror ("cannot bind to address");
    exit (-1);
}
i_addrlen = sizeof (i_addr);

if (listen (s, 3) == -1) {
   perror ("listen failed");
   exit (-1);
}
```

Hints (5)

```
fd_set working_set, base_set;
FD_SET(sd, &base_set);

while(1)
{    ....copy the base to working set...
     rc_select = select (sd + 1, &working_set, NULL, NULL, &select_timeout);
     /* Check to see if the select call failed. */
     if (rc_select < 0) {
       perror("select() failed");
       check error status and act accordingly
     }
     /* Check to see if the 'n' minute time out expired.       */
     if (rc_select == 0) {
       fprintf(stderr, "\n select() timed out. \n");
       return -1;
     }
     /* Check to see if there is a incoming connection request  */
     if (FD_ISSET(sd, &working_set))
     {....act based on the set descriptor
```